10 Средства межпроцессного взаимодействия (IPC)

10.1 Общие сведения

IPC – ***I***nter-***P***rocess ***C***ommunications

Основные группы IPC:

Сигнальные – передача информации (извещений) об однократных событиях, происходящий в произвольные моменты времени, в т.ч. об ошибках: прерывания (исключения), сообщения, сигналы.
В Unix-системах – ***сигналы*** (***signal***), были рассмотрены выше

Канальные – передача потока (stream) неструктурированных данных: ***каналы*** (***транспортеры***, ***pipe***, ***FIFO***), ***сокеты*** потокового типа и т.п.

Очереди сообщений (***message queue***) – передача данных в виде законченных фрагментов с определенной структурой и в определенном порядке: разнообразные ***MQ***, ***сокеты*** датаграмного типа. Промежуточное положение между канальными IPC и одиночными сообщениями

Разделяемая память (***shared memory***) – параллельный доступ нескольких процессов к одной и той же области памяти, обычно на основе отображения страниц памяти в несколько виртуальных адресных пространств.

Объекты синхронизации (***I***nterprocess ***S***ynchronization ***O***bjects, ***ISO***) – передача информации о событиях (наступлении условий, изменении состояний), т.е. отчасти пересекаются роли с сигнальными IPC: ***семафоры*** (***semaphore***), ***мьютексы*** (***mutex***), ***события*** (***event***), ***барьеры*** и т.п.

В Unix-системах – группы IPC, стремящихся закрыть весь спектр задач.
Отдельно – сокеты и специфические IPC потоков (позже)

10.2 Каналы: pipe и FIFO

Типичные каналы. Как и для всех каналов, доступ максимально близок к обычным файлам.

Основные особенности по сравнению с файлами:
– последовательный доступ (чтение и запись)
– разрушающее чтение

Обеспечивается сохранение последовательности данных в потоке

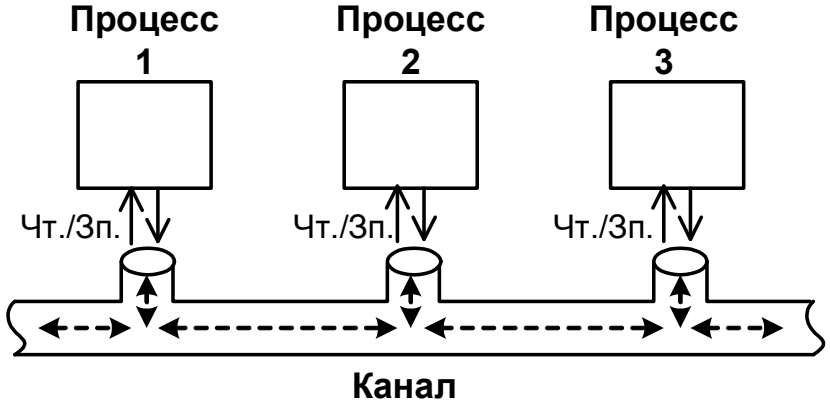

Рис. – Канал

Основные проблемы:
– нет структурирования данных в потоке, никак не контролируется соответствие «порций» записываемых и считываемых данных
– нет идентификации ни отправителя, ни получателя данных
– нет механизмов явного определения состояния взаимодействующих процессов

Как следствие, коллизии доступа при использовании каналов. Необходимость их предотвращения и/или разрешения.

Несмотря на недостатки, один из наиболее простых и удобных в использовании, поэтому наиболее используемых IPC-объектов. Унификация доступа с файлами и устройствами.

10.2.1 Неименованные каналы

Традиционное название – ***pipe***. Существуют с ранних версий Unix как один из базовых механизмов взаимодействия.

Идентификация: только ***дескриптор*** **`fd`**
Как следствие, не может быть открыт произвольным процессом в произвольный момент времени, но наследуется порожденными процессами от родительского процесса, поэтому может использоваться только между процессами-«родственниками».

Данные сохраняются в канале только пока существует хотя бы один дескриптор, связанный с каналом. Канал без дескрипторов (без процессов-пользователей) удаляется со всеми остающимися в нем данными.

Особенность: пары дескрипторов для доступа (чтение и запись)

Уровень командной строки:

```
<команда 1> | <команда 2>
```

Уровень системных вызовов:

```
int pipe( int fds[2]);
…
read( … fds[0] … );
…
write( … fds[1] … );
```

Особенности поведения при отсутствии при отсутствии данных для считывания, при частично закрытых дескрипторах.

10.2.2. Именованные каналы

Традиционное название – ***FIFO***. Появились в BSD Unix (однако там же получили и более функционального конкурента – сокеты).

Идентификация:
– в файловой системе – ***имя*** (без ограничений)
– в роцессе после открытия – ***дескриптор*** **`fd`** (режимы чтения, записи, чтения и записи)

Именованный канал с точки зрения файловой системы – обычный файл с особенностями доступа. Он может быть открыт произвольным процессом в произвольный момент времени (наследовать дескрипторы тоже можно), поэтому может использоваться любыми процессами.

Данные в файле FIFO сохраняются пока не будут прочитаны (или пока файл не будет явным образом удален).

Уровень командной строки:

```
mkfifo [опции] имя
```

Уровень системных вызовов:

```
int mkfifo( const char *pathname, mode_t mode)
int mknod( ... )
…
open( pathname … )
…
read( … fds[0] … );
…
write( … fds[1] … );
```

10.3 System V IPC (sVipc)

Стремящийся к функциональной полноте набор объектов IPC, призванный решать почти все типичные задачи взаимодействия (в дополнение к «базовым» IPC: сигналам и каналам).

«Старые» IPC, однако поддержка сохраняется в большинстве систем, а некоторые возможности уникальны по сравнению с POSIX IPC.

Особенность – двухстадийная идентификация:
– «ключ» – генерируется функцией **`ftok()`** из файла/inode (идентифицируемого именем) и произвольно выбранного целого числа (часто называемого «номером проекта»);
– идентификатор ID – возвращается функциями **`***get()`** при создании конкретного объекта.

```
key_t ftok( const char* fname, int proj_id)
```

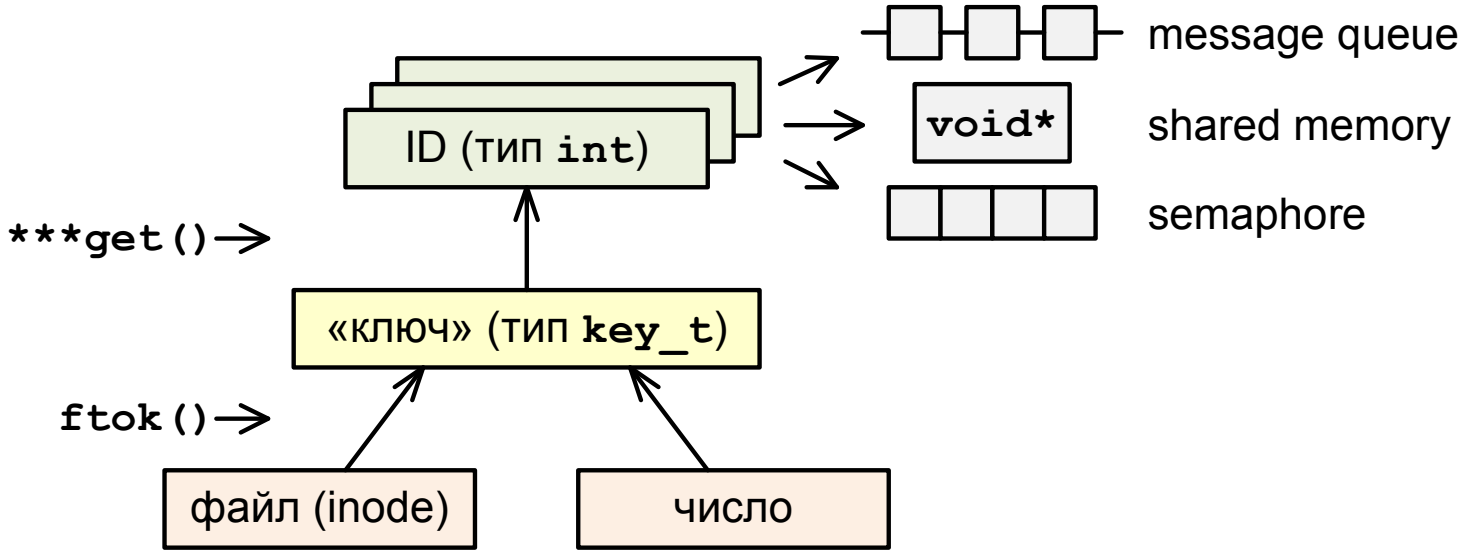

Рис. – Идентификация sVipc

Ключи и идентификаторы глобальны в системе. Одинаковые ключи гарантируют получение одних и тех же идентификаторов. Пространства идентификторов объектов разного типа не пересекаются.

Наличие промежуточного идентификатора – возможность совместного использования различных объектов в рамках комплексного решения

Функции для работы с конкретными объектами также по возможности однотипны, некоторые из них общие.
Для всех объектов определены функции **`***get()`** и **`***ctl()`**.
Функции **`***ctl()`** – управление объектом, поддерживаются различные операции, среди которых обязательно есть **`IPC_RMID`**, **`IPC_STAT`** и **`IPC_SET`**.

Общая особенность – глобальность объектов, сохранение их состояния независимо от процессов-пользователей. Например, разделяемая память продолжает расходовать ресурсы после отключения всех процессов, а семафор остается заблокированным для новых пользователей.

10.3.1 Семафоры sVipc

Многозначные векторные – массив счетчиков. Особенность – возможность выполнения последовательностей операций над вектором семафоров одним системным вызовом.

Рис. – «векторные» семафоры sVipc

Функции API:

```
semget( key_t key, int dim, int flags);
semctl( int sem_id, struct sembuf* ops, size_t n);
sem_ctl( int sem_id, int sem_num, int cmd, …)
```

Операции над семафором:

```
struct sembuf {
  short sem_num;
  short sem_op;
  ushort sem_flg;
}
```

Коды операций sem_op:

`op` > 0 – безусловное увеличение счетчика на величину **`op`**

`op` < 0 – условное (с ожиданием) уменьшение счетчика на величину |**`op`**|

`op` = 0 – ожидание обнуления счетчика

Флаги:

`SEM_UNDO` – откат операции при завершении процесса

10.3.2 Разделяемая память sVipc

Обычная разделяемая память с промежуточной стадией объекта IPC, который должен быть «отображен» в адресное пространство.

Рис. – Разделяемая память sVipc

Функции API:

```
shmget( key_t key, size_t size, int flags);
shmat( int shmid, char* addr, int flags);
shmdt( char* addr);
shmctl( int shmid, int cmd, struct shmid_ds *buf);
```

10.3.3 Очереди сообщений sVipc

Связный список сообщений

Структура сообщения:
– тип сообщения (long int)
– длина информационной части
– блок данных (необязательный)

Функции:

```
msgget( key_t key, int msgflg);
msgsnd( int msq_id,
  const void *msgp, size_t msg_sz,
  int flags);
msgrcv( int msq_id,
  void *msgp, size_t msg_sz, long msg_typ,
  int flags);
msgctl( int msq_id, int cmd, struct msqid_ds *buf)
```

Действия при приеме в зависимости от значения «заказанного» типа:

msg_typ **> 0** – первое по порядку сообщение этого типа

msg_typ **< 0** – первое по порядку сообщение с кодом типа не более |*msg_typ*|

msg_typ **= 0** – первое по порядку сообщение независимо от типа

10.4 POSIX IPC

Набор, похожий на sVipc, также претендующий на функциональную полноту

10.4.1 Семафоры POSIX

Обычный классический семафор с обычным набором поддерживаемых операций

Идентификация:
– имя (по правилам файловой системы)
– тип **sem_t**

```
sem_t* sem_open( const char* sem_name, int oflags)
sem_t* sem_open( const char* sem_name, int oflags,
mode_t mode, uint val)
sem_close()
sem_unlink()
int sem_getvalue()
```

```
int sem_post()
int sem_wait()
int sem_trywait()
int sem_timedwait()
```

Безымянные семафоры – для использования потоками одного процесса (группы потоков). Вызовы **`sem_open()`**, **`sem_close()`**, **`sem_unlink()`** недействительны, вместо них используются:

```
sem_init()
sem_destroy()
```

Принципиальная возможность использовать безымянные семафоры между процессами сохраняется, но для этого их необходимо размещать в разделяемой памяти (POSIX или System V).

10.4.2 Разделяемая память POSIX

Реализуется через общий механизм отображения файлов в память: функция **`mmap()`**.

Интегрировано с общим механизмом отображения дискового пространства в память.

Идентификация объектов (блоков) «разделяемая память»:
– имя (отображаемого файла)
– целочисленный дескриптор (из пространства fd)

Создание/открытие, закрытие и удаление (поскольку дескриптор объекта совместим с обычными файловыми, для его закрытия служит универсальный вызов):

```
shm_open()
close()
shm_unlink()
```

К созданному объекту «разделяемая память» применяются функции работы с отображениями:

`mmap()`
`munmap()`
`mremap()`

После успешного отображения объекту «разделяемая память» функцией **`mmap()`** его дескриптор можно закрыть.

Особенность – необходимость своевременной «синхронизации» содержимого памяти разных процессов:

`msynk()`

«Блокировка» – запрет выгрузки содержимого в файл подкачки:

`mlock()`
`munlock()`

10.4.3 Очереди сообщений POSIX

Идентификация очереди:
– имя (по правилам файловой системы)
– тип **`mqd_t`**

Структура сообщения:
1) приоритет (тип) сообщения – unsigned
2) длина блока данных (может быть нулевой)
3) данные (если есть)

Создание/открытие, закрытие, удаление очереди:

`mq_open()`
`mq_close()`
`mq_unlink()`

Типичное использование очереди:

`mq_send()`
`mq_receive()`

Дополнительная возможность – события. Процесс может «зарегистрироваваться» на события очереди:

`mq_notify()`

Доступные реакции на события:
– игнорирование
– генерация сигнала
– автоматический запуск потока с обработчиком

Отдельные функции для работы с атрибутами:

`mq_getattr()`
`mq_setattr()`